

# Numbers and Strings

(<http://docs.oracle.com/javase/tutorial/java/data/index.html>)

(<https://docs.oracle.com/javase/8/docs/>)

(<http://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html>)

# Objectives

- **Working with Numbers:**
  - Wrapper classes: Number, Character
  - Auto boxing and unboxing .
- **The `java.lang.Math`**
- **String class:**
  - Create and manipulate strings.
  - Compares the String and StringBuilder classes.
- **Scanning Text**
- **Formatting output**

# Introduction

- A class can contain no data field or some data fields
- Some operations on numbers are critical such as converting a string to number, ...
- Java libraries have classes which wrap a number (primitive type) in it and support operations on numbers. They are called as wrapper classes.
- String is a common data type and it is a pre-defined class in Java library.
- The **java.lang** package contains all of them

# Numbers Classes

- Java platform provides *wrapper* classes for each of the primitive data types.
- `java.lang.Object`
  - `java.lang.Boolean` (implements `java.lang.Comparable<T>`, `java.io.Serializable`)
  - `java.lang.Character` (implements `java.lang.Comparable<T>`, `java.io.Serializable`)
  - `java.lang.Character.Subset`
    - `java.lang.Character.UnicodeBlock`
  - `java.lang.Math`
  - `java.lang.Number` (implements `java.io.Serializable`)
    - `java.lang.Byte` (implements `java.lang.Comparable<T>`)
    - `java.lang.Double` (implements `java.lang.Comparable<T>`)
    - `java.lang.Float` (implements `java.lang.Comparable<T>`)
    - `java.lang.Integer` (implements `java.lang.Comparable<T>`)
    - `java.lang.Long` (implements `java.lang.Comparable<T>`)
    - `java.lang.Short` (implements `java.lang.Comparable<T>`)

All of the numeric wrapper classes are subclasses of the abstract class `Number`.

# Numbers Classes: A Declaration

**public final class Integer extends Number implements Comparable<Integer>**

## Fields

Modifier and Type	Field and Description
static int	<b>BYTES</b> The number of bytes used to represent a int value in two's complement binary form.
static int	<b>MAX_VALUE</b> A constant holding the maximum value an int can have, $2^{31}-1$ .
static int	<b>MIN_VALUE</b> A constant holding the minimum value an int can have, $-2^{31}$ .
static int	<b>SIZE</b> The number of bits used to represent an int value in two's complement binary form.
static <b>Class&lt;Integer&gt;</b>	<b>TYPE</b> The Class instance representing the primitive type int.

# Numbers Classes: A Declaration

**public final class Integer extends Number implements Comparable<Integer>**

We can not create a sub-class of a wrapper class

## Constructors

### Constructor and Description

**Integer**(int value)

Constructs a newly allocated Integer object that represents the specified int value.

**Integer**(String s)

Constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

# Auto boxing and Unboxing

- Java 5.0 introduces two very simple but convenient functions that unwrap wrapper objects and wrap up primitives.
- Converting a primitive value into an object of the corresponding wrapper class is called auto boxing.
- Converting an object of a wrapper type to its corresponding primitive value is called unboxing.

# Auto boxing and Unboxing

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double



# Example 1

```

1
2 package session06;
3 import java.lang.Integer;
4 public class WrapperExample1 {
5     public static void main(String args[]) {
6         // Đổi int thành Integer
7         int a = 20;
8         Integer i = new Integer(10);
9         Integer k = new Integer("50");
10        Integer j = a + i; // autoboxing, tự động đổi int thành Integer trong nội bộ trình biên dịch
11        Integer m = a + k;
12        System.out.println("Value of j: " + j);
13        System.out.println("Value of j: " + m);
14    }
15 }
16
17
18

```

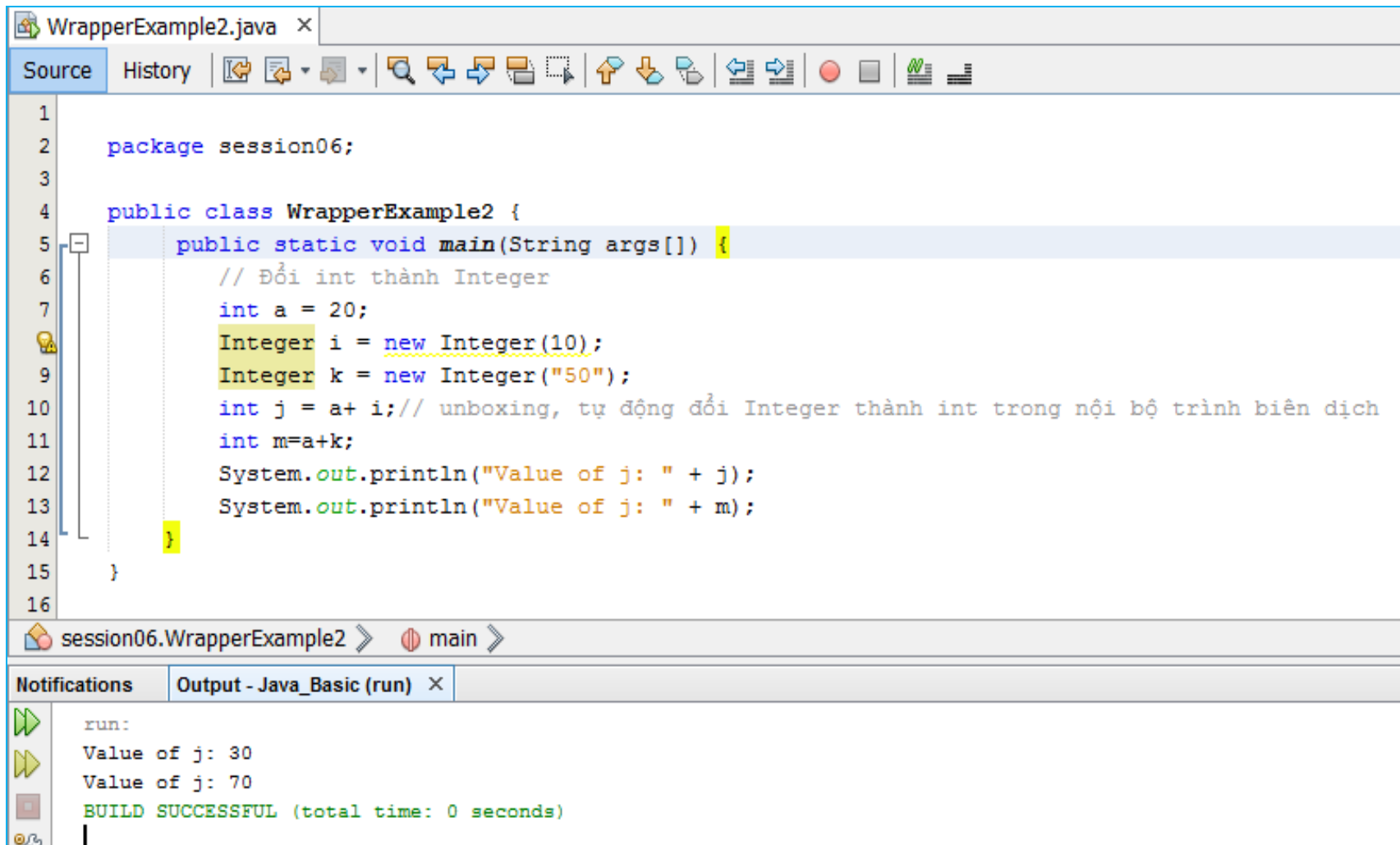
Notifications | Output - Java\_Basic (run) ×

```

run:
Value of j: 30
Value of j: 70
BUILD SUCCESSFUL (total time: 0 seconds)

```

# Example 2



```

1
2 package session06;
3
4 public class WrapperExample2 {
5     public static void main(String args[]) {
6         // Đổi int thành Integer
7         int a = 20;
8         Integer i = new Integer(10);
9         Integer k = new Integer("50");
10        int j = a + i; // unboxing, tự động đổi Integer thành int trong nội bộ trình biên dịch
11        int m = a + k;
12        System.out.println("Value of j: " + j);
13        System.out.println("Value of j: " + m);
14    }
15 }
16

```

session06.WrapperExample2 > main >

Notifications Output - Java\_Basic (run) ×

```

run:
Value of j: 30
Value of j: 70
BUILD SUCCESSFUL (total time: 0 seconds)

```

# Numbers Classes: A Declaration

**public final class Integer extends Number implements Comparable<Integer>**

Some  
common  
methods

Wrapper  
classes are  
immutable  
(non-  
changeable)  
because they  
do not have  
setters

```
byte, short, ...    byteValue(), shortValue(), intValue(), longValue(),
                    floatValue(), doubleValue()

static int          compare(int x, int y)

int                compareTo(Integer anotherInt)

static int          compareUnsigned(int x, int y)

boolean            equals(Object obj)

static Integer      getInteger(String nm), getInteger(String nm, int val),
                    getInteger(String nm, Integer val)

static int          lowestOneBit(int i)

static int          max(int a, int b), min(int a, int b)

static int          parseInt(String s), parseInt(String s, int radix)
                    parseUnsignedInt(String s), parseUnsignedInt(String s, int radix)

static String       toBinaryString(int i), toHexString(int i), toOctalString(int i)
                    toString(int i), toString(int i, int radix), toUnsignedString(int i)

String             toString()

static long         toUnsignedLong(int x)

static String       toUnsignedString(int i, int radix)

static Integer      valueOf(int i), valueOf(String s), valueOf(String s, int radix)
```

# Numbers Classes...

- We use a Number object rather than a primitive when:
  - As an argument of a method that expects an object.
  - To use constants defined by the class, such as `MIN_VALUE` and `MAX_VALUE`.
  - To use class methods for `converting` values to and from other primitive types.

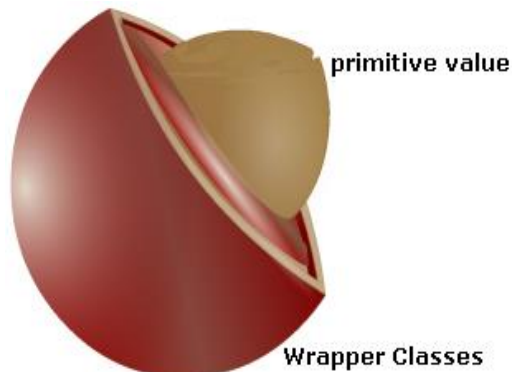
# Numbers Classes...

```
public class MainClass{
    public static void main(String[] argv){
        System.out.println(Integer.MAX_VALUE);
    }
}
```

2147483647

# Numbers Classes- A Demo

- java.lang.Object
  - java.lang.Boolean
  - java.lang.Character
  - java.lang.Number
    - java.lang.Byte
    - java.lang.Double
    - java.lang.Float
    - java.lang.Integer
    - java.lang.Long
    - java.lang.Short



```

WrapperExample3.java
Source History
1
2 package session06;
3
4 public class WrapperExample3 {
5     public static void main(String args[]) {
6         int n,a,b,c;
7         a=20;
8         n=Integer.parseInt("30");
9         b=a+n;
10        c=Integer.parseInt("1A",16);
11
12        System.out.println("Value of b : "+ b);
13        System.out.println("Value of c : "+ c);
14
15    }
16 }
17
18
session06.WrapperExample3
Notifications Output - Java_Basic (run) X
run:
Value of b : 50
Value of c : 26
BUILD SUCCESSFUL (total ti

```

Boxing/auto boxing:  
encapsulating/wrapping a  
primitive value to an object.  
Unboxing: get primitive value  
wrapped in a wrapper object.

# Numbers Classes- A Demo

The screenshot shows an IDE window titled 'WrapperExample4.java'. The code is as follows:

```

3
4 import java.util.Scanner;
5
6 public class WrapperExample4 {
7
8     public static void main(String args[]){
9         int money;
10        float RateOfInterest;
11        float Totalmoney;
12        Scanner sc=new Scanner(System.in);
13
14        System.out.println("Enter your money:" );
15        money=Integer.parseInt( sc.nextLine());
16
17        System.out.println("Enter RateOfInterest:" );
18        RateOfInterest=Float.parseFloat(sc.nextLine());
19        Totalmoney=money*RateOfInterest;
20
21        System.out.println("Total of your RateOfInterest money :" + Totalmoney);
22    }
23 }
24

```

Below the code editor is the 'Output - Java\_Basic (run)' window, which displays the following text:

```

run:
Enter your money:
300000
Enter RateOfInterest:
0.8
Total of your RateOfInterest money :240000.0
BUILD SUCCESSFUL (total time: 7 seconds)

```

# Numbers Classes- A Demo

```

1
2 import java.util.Scanner;
3
4
5 public class Person {
6
7     public static void main(String[] args) {
8         Scanner sc=new Scanner(System.in);
9         String fullname,address;
10        int age;
11
12        System.out.println("Enter your fullname:");
13        fullname=sc.nextLine();
14        System.out.println("Enter your age:");
15        age=sc.nextInt();
16        System.out.println("Enter Your Address:");
17        address=sc.nextLine();
18
19        System.out.println("Information : " + fullname + "-" + age + "-" + address);
20
21    }
22
23 }
```



# Numbers Classes- A Demo

Output - Chapter05 (run)

```
run:
Enter your fullname:
    Tran Hoai Bao
Enter your age:
    20
Enter Your Address:
Information : Tran Hoai Bao-20-
BUILD SUCCESSFUL (total time: 9 seconds)
```

# Numbers Classes- A Demo

```

Person.java x
Source History
1
2 import java.util.Scanner;
3
4
5 public class Person {
6
7     public static void main(String[] args) {
8         Scanner sc=new Scanner(System.in);
9         String fullname,address;
10        int age;
11
12        System.out.println("Enter your fullname:");
13        fullname=sc.nextLine();
14        System.out.println("Enter your age:");
15        age=Integer.parseInt(sc.nextLine());
16        System.out.println("Enter Your Address:");
17        address=sc.nextLine();
18
19        System.out.println("Information :" + fullname + "-" +age + "-" + address);
20
21    }
22
23 }

```

# Numbers Classes- A Demo

Output - Chapter05 (run)



run:

Enter your fullname:

Nguyen Hoai Bao

Enter your age:

20

Enter Your Address:

D1 Khu Cong Nghe Cao Q9 HCM

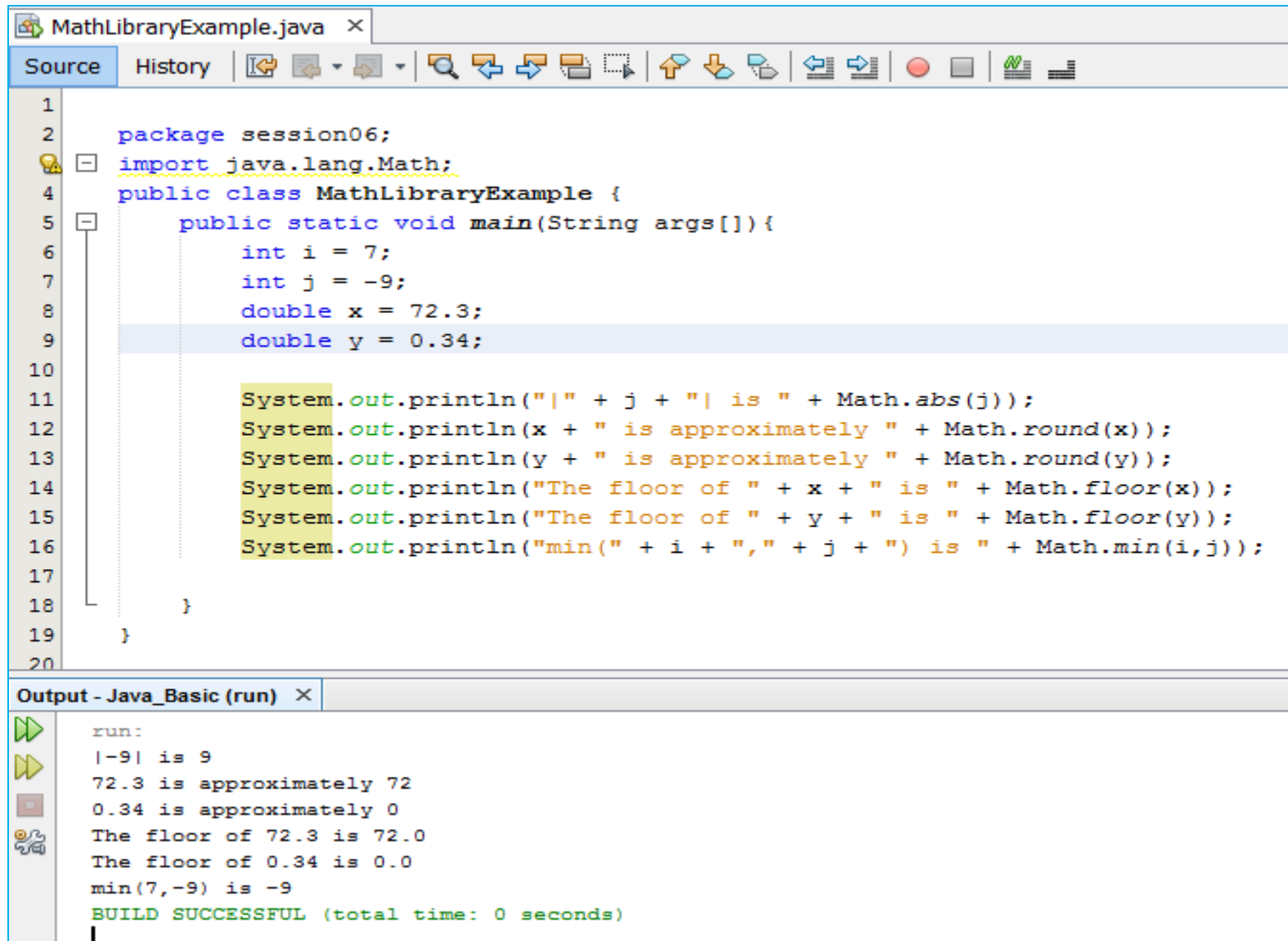
Information :Nguyen Hoai Bao-20-D1 Khu Cong Nghe Cao Q9 HCM

BUILD SUCCESSFUL (total time: 27 seconds)

# The `java.lang.Math` class

- The `Math` class in the `java.lang` package provides methods and constants for doing more advanced mathematical computation, including:
  - **Constants and Basic Methods:** `Math.E`, `Math.PI`,...
  - **Basic static methods:** `ceil(double d)`, `floor(double d)`, `abs(int i)`...
  - **Exponential and Logarithmic Methods:** `exp(double d)`, `sqrt(double d)`, `pow(double base, double exponent)`
  - **Trigonometric Methods:** `cos(double d)`, `sin(double d)`
  - **Random Numbers:** The `random()` method returns a pseudo-randomly selected number between 0.0 and 1.0.

# Demo Java.lang.Math



The screenshot shows an IDE window titled "MathLibraryExample.java". The code defines a package "session06", imports "java.lang.Math", and creates a class "MathLibraryExample" with a "main" method. Inside "main", variables are declared: "int i = 7", "int j = -9", "double x = 72.3", and "double y = 0.34". The program then prints several results using "System.out.println": the absolute value of j, the rounded value of x, the rounded value of y, the floor of x, the floor of y, and the minimum of i and j. The output window at the bottom shows the execution results, confirming the calculations.

```

1
2 package session06;
3 import java.lang.Math;
4 public class MathLibraryExample {
5     public static void main(String args[]){
6         int i = 7;
7         int j = -9;
8         double x = 72.3;
9         double y = 0.34;
10
11         System.out.println("|" + j + "| is " + Math.abs(j));
12         System.out.println(x + " is approximately " + Math.round(x));
13         System.out.println(y + " is approximately " + Math.round(y));
14         System.out.println("The floor of " + x + " is " + Math.floor(x));
15         System.out.println("The floor of " + y + " is " + Math.floor(y));
16         System.out.println("min(" + i + ", " + j + ") is " + Math.min(i,j));
17     }
18 }
19 }
20

```

Output - Java\_Basic (run) X

```

run:
|-9| is 9
72.3 is approximately 72
0.34 is approximately 0
The floor of 72.3 is 72.0
The floor of 0.34 is 0.0
min(7,-9) is -9
BUILD SUCCESSFUL (total time: 0 seconds)

```

# Characters

- Unicode character, 2 bytes
- Character class also offers a number of useful class (i.e., static) methods for manipulating characters.
- `Character ch = new Character('a');`
- Some methods in this class
  - `boolean isLetter(char ch)/ isDigit(char ch)/ isUpperCase(char ch)`
  - `char toUpperCase(char ch) ...`
- A character preceded by a backslash (`\`) is an *escape sequence* and has special meaning to the compiler.

# Characters - Demo

The screenshot shows an IDE window titled 'TestCharacters.java'. The code defines a package 'session06' and a public class 'TestCharacters' with a 'main' method. The 'main' method prints a message and the values of two character variables, 'ch1' and 'ch2'. The output window at the bottom shows the execution results, confirming the values of 'ch1' and 'ch2' and indicating a successful build.

```

1
2  package session06;
3
4  public class TestCharacters {
5
6      public static void main(String args[]) {
7          System.out.println("She said \"Hello!\" to me.");
8          Character ch1 = 'a';
9          Character ch2 = 65;
10         System.out.println("Value of char variable ch1 is : " + ch1);
11         System.out.println("Value of char variable ch2 is : " + ch2);
12     }
13 }
14

```

Output - Java\_Basic (run) ×

```

run:
She said "Hello!" to me.
Value of char variable ch1 is :a
Value of char variable ch2 is :A
BUILD SUCCESSFUL (total time: 0 seconds)

```

# Strings

- Java uses the **String**, **StringBuffer**, and **StringBuilder** classes to encapsulate strings of characters (16-bit Unicode).

java.lang.Object

java.lang.String (implements java.lang.CharSequence,  
java.lang.Comparable<T>, java.io.Serializable)

java.lang.StringBuffer (implements java.lang.CharSequence,  
java.io.Serializable)

java.lang.StringBuilder (implements  
java.lang.CharSequence, java.io.Serializable)

Interface **Serializable** declared methods for processing a string as a stream of characters (write string to file, ...)



# The *String* Class

- The String class contains an **immutable** string (Once an instance is created, the string it contains cannot be changed) ← **No setter is implemented**
- **Almost of it's methods will return a new string.**
- Construct a string:

`String s1 = new String("immutable");`

`String s2= new String (new char[] {'a', 'b', 'c'});`

or

`String s3 = "immutable";`

# String pool

```
public class StringDemo {
    public static void main (String[] args)
    {
        String s1="Hello"; // string pool
        String s2="Hello"; // string pool
        System.out.println("s1==s2: " + (s1==s2));
        String s3= new String("Hello");
        String s4= new String("Hello");
        System.out.println("s3==s4: " + (s3==s4));
        System.out.println("s3 equals s4: " + (s3.equals(s4)));
        String s5= new String ( new char[] { 'H', 'E', 'L', 'L', 'O' });
        System.out.println("s3 equals s5 ignoring case: " + (s3.equalsIgnoreCase(s5)));
        System.out.println(s5);
        s5= s5.toLowerCase();
        System.out.println(s5);
    }
}
```

String pool

Hello

s1

s2

Shallow comparing: Compare two references

Deep comparing: Compare two values

hello

HELLO

HELLO

garbage

s5

s5

String pool: a way to save memory

Output - Chapter08 (run)

```
run:
s1==s2: true
s3==s4: false
s3 equals s4: true
s3 equals s5 ignoring case: true
HELLO
hello
```

# The *String* Class

Compare 2 strings: should use equals()

```
String st1 = "abc";
String st2 = "xyz";
if(st1.equals(st2)){
    ...
}
```

# The *String* Class

Modifier and Type	Method and Description
char	<a href="#"><u>charAt</u></a> (int index)
char[]	<a href="#"><u>toCharArray</u></a> ()
byte[]	<a href="#"><u>getBytes</u></a> ()
int	<a href="#"><u>codePointAt</u></a> (int index), <a href="#"><u>compareTo</u></a> ( <a href="#"><u>String</u></a> anotherString) <a href="#"><u>compareToIgnoreCase</u></a> ( <a href="#"><u>String</u></a> str), <a href="#"><u>hashCode</u></a> (), <a href="#"><u>indexOf</u></a> (int ch), <a href="#"><u>indexOf</u></a> (...), <a href="#"><u>lastIndexOf</u></a> (...), <a href="#"><u>length</u></a> ()
<a href="#"><u>String</u></a>	<a href="#"><u>trim</u></a> (), <a href="#"><u>toString</u></a> (), <a href="#"><u>concat</u></a> ( <a href="#"><u>String</u></a> str), <a href="#"><u>replace</u></a> (...), <a href="#"><u>replaceAll</u></a> (...) <a href="#"><u>replaceFirst</u></a> (...), <a href="#"><u>substring</u></a> (...), <a href="#"><u>toLowerCase</u></a> (...), <a href="#"><u>toUpperCase</u></a> (...)
static <a href="#"><u>String</u></a>	<a href="#"><u>copyValueOf</u></a> (...), <a href="#"><u>format</u></a> (...), <a href="#"><u>valueOf</u></a> (...)
boolean	<a href="#"><u>contains</u></a> ( <a href="#"><u>CharSequence</u></a> s), <a href="#"><u>endsWith</u></a> ( <a href="#"><u>String</u></a> suffix), <a href="#"><u>startsWith</u></a> (...), <a href="#"><u>isEmpty</u></a> (), <a href="#"><u>matches</u></a> ( <a href="#"><u>String</u></a> regex) <a href="#"><u>equals</u></a> ( <a href="#"><u>Object</u></a> anObject), <a href="#"><u>equalsIgnoreCase</u></a> (...)
void	<a href="#"><u>getChars</u></a> (int srcBegin, int srcEnd, char[] dst, int dstBegin)
<a href="#"><u>String</u></a> []	<a href="#"><u>split</u></a> ( <a href="#"><u>String</u></a> regex), <a href="#"><u>split</u></a> ( <a href="#"><u>String</u></a> regex, int limit)
<a href="#"><u>CharSequence</u></a>	<a href="#"><u>subSequence</u></a> (int beginIndex, int endIndex)

# The *String* Class

```
import java.util.Scanner;

public class StringDemo {

    public static void main(String aegs[]){
        Scanner sc= new Scanner(System.in);
        String origin, replaced, replacement;
        System.out.print("Enter original string:");
        origin= sc.nextLine();
        System.out.print("Enter replaced string:");
        replaced= sc.nextLine();
        System.out.print("Enter replacing string:");
        replacement= sc.nextLine();
        origin = origin.replaceAll(replaced, replacement);
        System.out.println("After replacing:" + origin);
        System.out.println("Uppercase:" + origin.toUpperCase());
        System.out.println("Origin:" + origin);
        System.out.print("Enter the index of extracted character:");
        int index= Integer.parseInt(sc.nextLine());
        System.out.format("The %d(th) character:%c\n", index, origin.charAt(index));
    }
}
```

```
run:
Enter original string:do not love me
Enter replaced string:love
Enter replacing string:hate
After replacing:do not hate me
Uppercase:DO NOT HATE ME
Origin:do not hate me
Enter the index of extracted character:3
The 3(th) character:n
```

# ***StringBuffer, StringBuilder Classes***

- Java's **StringBuffer** and **StringBuilder** classes represent strings that can be dynamically modified.
  - StringBuffer is threadsafe.
  - StringBuilder (introduced in 5.0) is not threadsafe.
- Almost of their methods are the same as methods in the String class.

**Thread:** Unit of code (method) is running

**Multi-threading program:** A program has some threads running concurrently. If 2 threads access common data, their values are not unpredictable. So, in multi-thread programming, JVM supports a mechanism in which accesses to common resources must carry out in sequence based on synchronized methods.

**Threadsafe class:** A class with synchronized methods.

# The *StringBuffer* - *threadsafe*

public final class **StringBuffer** extends [Object](#)  
implements [Serializable](#), [CharSequence](#)

```
public class StringBufferDemo {
    public static void main(String args[]){
        StringBuffer sBuf= new StringBuffer ("01234567");
        System.out.println(sBuf);
        sBuf.append("ABC");
        System.out.println(sBuf);
        sBuf.insert(2, "FAT PERSON");
        System.out.println(sBuf);
        sBuf.reverse();
        System.out.println(sBuf);
    }
}
```

run:

01234567

01234567ABC

01FAT PERSON234567ABC

CEA765432NOSREP TAF10

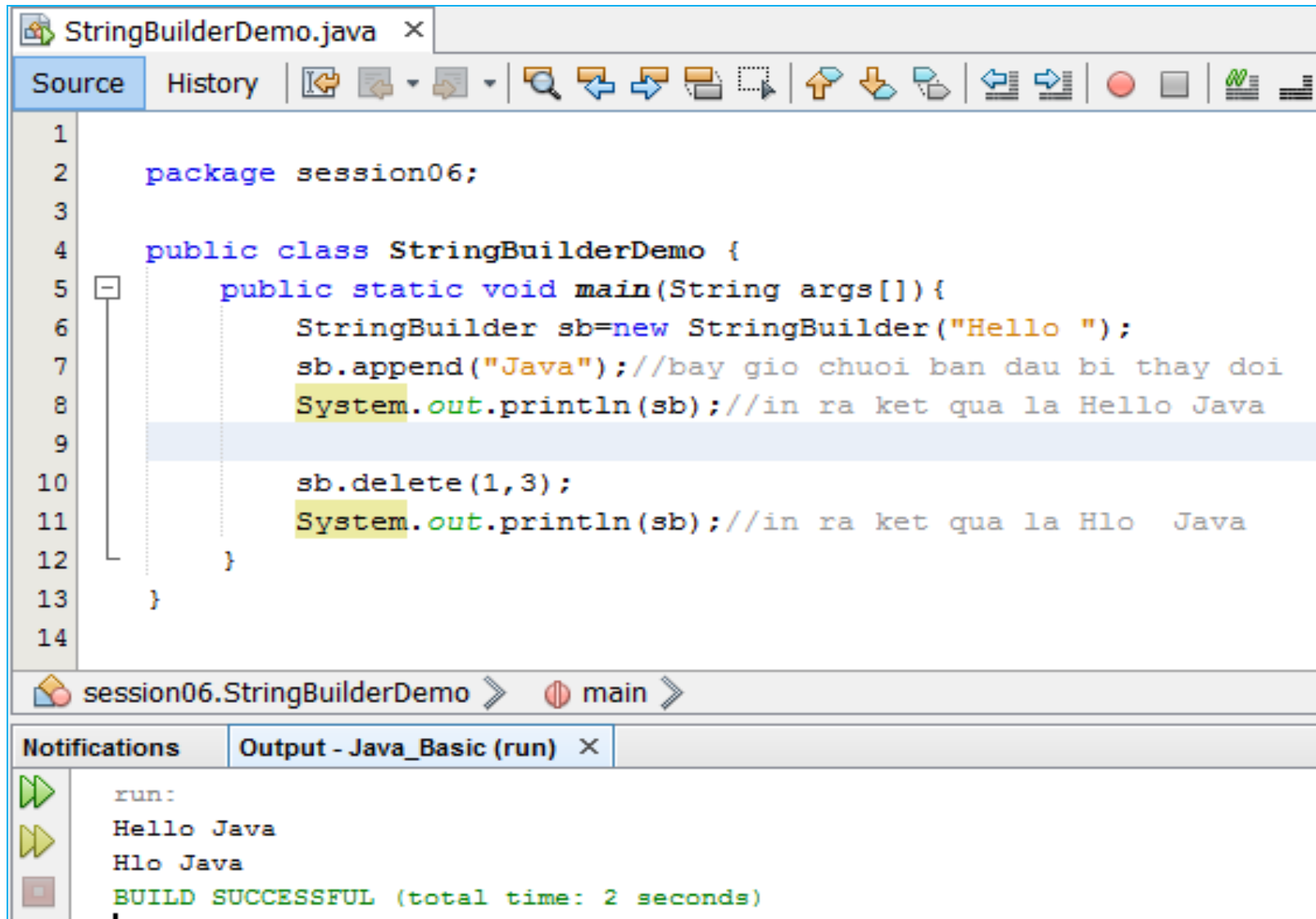
# ***StringBuilder***

public final class **StringBuilder** extends [Object](#)  
implements [Serializable](#), [CharSequence](#)

- The StringBuilder class was introduced in 5.0. It is nearly identical to StringBuffer.
- Major difference: string builders are **not threadsafe**.
- If you want multiple threads to have **concurrent access** to a mutable string, use a string buffer.
- If your mutable string will be accessed only by a single thread, there is an advantage to using a string builder, which will generally execute faster than a string buffer.



# StringBuilder



```

1
2  package session06;
3
4  public class StringBuilderDemo {
5      public static void main(String args[]){
6          StringBuilder sb=new StringBuilder("Hello ");
7          sb.append("Java");//bay gio chuoai ban dau bi thay doi
8          System.out.println(sb);//in ra ket qua la Hello Java
9
10         sb.delete(1,3);
11         System.out.println(sb);//in ra ket qua la Hlo  Java
12     }
13 }
14

```

session06.StringBuilderDemo > main >

Notifications    Output - Java\_Basic (run) ×

```

run:
Hello Java
Hlo Java
BUILD SUCCESSFUL (total time: 2 seconds)

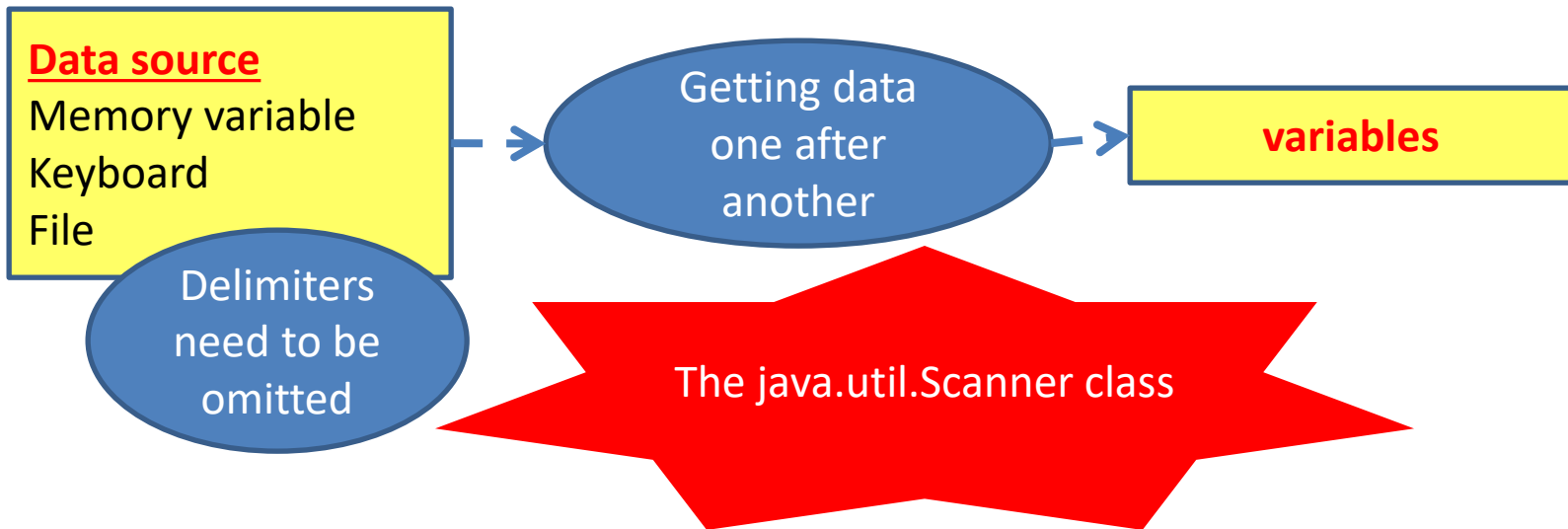
```

# String Concatenation, the Easy Way

- 02 ways:
  - `String.concat()` method of the `String` class and the `StringBuffer.append()`.
  - Overloaded `+` operator.

# Scanning Text

How to get data from a data source?



- **Class:** `java.util.Scanner`
- **Data in data source are characters**
- **Methods for getting data:** `next()`, `nextXXX()`
- **Methods for checking availability of data :** `hasXXX()`
- **Token:** group of characters that has a meaning.

# Scanning data from a string

```
import java.util.Scanner;

public class ScannerDemo {
    public static void main(String[] args) {
        String S= "Anh hùng khôn quá mỹ nhân quan";
        Scanner sc1= new Scanner (S);
        while (sc1.hasNext()) System.out.println(sc1.next());
        System.out.println();
        String S2= "abc 123 556.78";
        Scanner sc2= new Scanner (S2);
        System.out.println(sc2.next());
        System.out.println(sc2.nextInt());
        System.out.println(sc2.nextDouble());
        System.out.println();
        String S3= "  How are      you!  ";
        String delim ="[au\\s]+"; // s:space, +: >=1 occurrence
        Scanner sc3= new Scanner (S3);
        sc3.useDelimiter(delim);
        while (sc3.hasNext()) System.out.println(sc3.next());
    }
}
```

**Output - Chapte**

run:

Anh  
hùng  
khôn  
quá  
mỹ  
nhân  
quan

abc  
123  
556.78

How  
re  
yo  
!

BUILD SUCC

The default delimiter is the blank character. You can designate delimiters.  
[au\\s] means that a, u and space(\s) are delimiters.  
+ means that number of occurrences is equal or greater than 1

# Scanning data from a string

```

1
2 package session05;
3
4 import java.util.Scanner;
5
6 public class DemoString {
7
8     public static void main(String[] args) {
9         Scanner sc;
10        String s = "I love you 4 so much. 34 I 23 want to marry you";
11        sc = new Scanner(s);
12        sc.useDelimiter("[[ .]\\d]+");
13        while (sc.hasNext()) {
14            String token = sc.next();
15            System.out.println(token);
16        }
17    }
18 }

```

```

Output - Java_Basic (run) x
run:
I
love
you
so
much
I
want
to
marry
you
BUILD SUCCESSFUL (total time: 0 seconds)

```

# Splitting a string into substrings

The method `split(delimiters)` of the class `String` and the `java.util.StringTokenizer` are used.

```
import java.util.StringTokenizer;
public class SplittingStringDemo {
    public static void main(String[] args) {
        String str = "I study hard. So, I pass the examination";
        // Using the method String[] split() of the String class
        String[] strs = str.split("[ ,.]+");
        for (String s:strs) System.out.println(s);
        System.out.println();
        // Using the java.util.StringTokenizer class
        StringTokenizer stk= new StringTokenizer(str,"[ ,.]");
        System.out.println("Number of substrings: " + stk.countTokens());
        while (stk.hasMoreTokens())
            System.out.println(stk.nextToken());
    }
}
```

Output - Chapter08 (run)

```
run:
I
study
hard
So
I
pass
the
examination

Number of substrings: 8
I
study
hard
So
I
pass
the
examination
BUILD SUCCESSFUL (total
```

# Formatting Output

%[argument\_index\$][flags][width][.precision]conversion

See API documentation for more details ([api/java/util/Formatter.html#syntax](http://api/java/util/Formatter.html#syntax)).

```
public class PrintWithFormat {
    public static void main (String[] args){
        String S="Hello";
        int i=5;
        long l=58;
        float f= 7.2f;
        double d=8.9;
        boolean b= true;
        char c='A';
        System.out.format ("%s,%2d,%4Xh,%5.2f,%10.3f,%3c,%b\n", S,i,l,f,d,c,b );
        System.out.format ("%3$3b,%1$3d,%2$12s\n", i,S,b);
    }
}
```

1 2 3

run:

```
Hello, 5,  3Ah, 7.20,      8.900,  A,true
true,  5,      Hello
```

# Formatting Output

```
import java.text.DecimalFormat;

public class DecimalFormatDemo {

    static public String customFormat(String pattern, double value ){
        DecimalFormat myFormatter = new DecimalFormat(pattern);
        String output = myFormatter.format(value);
        return output;
    }

    static public void main(String[] args) {
        System.out.println(customFormat("###,###.###", 123456.789));
        System.out.println(customFormat("###.##", 123456.789));
        System.out.println(customFormat("000000.000", 123.78));
        System.out.println(customFormat("$###,###.###", 12345.67));
    }
}
```

run:

123,456.789

123456.79

000123.780

\$12,345.67



# Summary

- **Working with Numbers:**
  - Wrapper classes: Number, Character
  - The java.lang.Math class
  - Autoboxing and unboxing .
- **The java.lang.Math**
- **String class:**
  - Create and manipulate strings.
  - Compares the String and StringBuilder classes.
- **Scanning Text**
- **Formatting output**